窗体顶端

# 数据结构与算法

您现在的位置：[希赛网](http://www.educity.cn/" \o "希赛网) > [云阅读](http://www.educity.cn/jiaocheng/) > [软件设计师考试试题分类精解（2018版）](http://www.educity.cn/jiaocheng/zt251.html" \t "http://www.educity.cn/jiaocheng/_blank) > **试题1(2017年下半年试题4)**

第 17 章：数据结构与算法应用作者：[希赛软考学院](http://www.educity.cn/jiaocheng/a%cf%a3%c8%fc%c8%ed%bf%bc%d1%a7%d4%ba.html" \o "希赛软考学院" \t "http://www.educity.cn/jiaocheng/_blank)    来源：希赛软考学院    2017年11月21日

# **试题1(2017年下半年试题4)**

阅读下列说明和C代码，回答问题 1 至问题 2，将解答写在答题纸的对应栏内。  
【说明】  
一个无向连通图G点上的哈密尔顿（Hamiltion）回路是指从图G上的某个顶点出发，经过图上所有其他顶点一次且仅一次，最后回到该顶点的路劲。一种求解无向图上哈密尔顿回路算法的基础私下如下：  
假设图G存在一个从顶点V0出发的哈密尔顿回路V1——V2——V3——...——Vn-1——V0。算法从顶点V0出发，访问该顶点的一个未被访问的邻接顶点V1，接着从顶点V1出发，访问V1一个未被访问的邻接顶点V2，..。；对顶点Vi，重复进行以下操作：访问Vi的一个未被访问的邻接接点Vi+1；若Vi的所有邻接顶点均已被访问，则返回到顶点Vi-1，考虑Vi-1的下一个未被访问的邻接顶点，仍记为Vi；知道找到一条哈密尔顿回路或者找不到哈密尔顿回路，算法结束。  
【C代码】  
下面是算法的C语言实现。  
（1）常量和变量说明  
n :图G中的顶点数  
c［］［］:图G的邻接矩阵  
K:统计变量，当期已经访问的定点数为k+1  
 x［k］:第k个访问的顶点编号，从0开始  
 Visited［x［k］］：第k个顶点的访问标志，0表示未访问，1表示已访问  
⑵C程序  
#include <stido.h>  
#include <stidb.h>  
#define MAX 100  
   
Vido Hamilton（int n,int x［MAX,int c[MAX][MAX]）｛  
in t ;  
in t visited[MAX];  
int k;  
  /\*初始化x数组贺visited数组\*/  
for （i=0:i<n;i++）｛  
      x[i]=0;  
      visited [i]=0;  
｝  
/\*访问起始顶点\*/  
k=0  
（  ）；  
x[0]=0  
K=k+1  
/\*访问其他顶点\*/  
while（k>=0）｛  
    x[k]=x[k]+1;  
    while（x[k]><n）｛  
         if （  ）&&c[x-[k-1]][x[k]＝1）｛/\*邻接顶点x[k]未被访问过\*/  
            break；  
         ｝else｛  
               x[k] = x[k] +1  
         ｝  
     ｝  
       if（x[k] <n-1&&（  ）｛ /\*找到一条哈密尔顿回路\*/  
           for （k=0;k<n;k++）｛  
                   prinf（〝%d--〝,x[k] ;  /\*输出哈密尔顿回路\*/  
           ｝  
           prinf（〝%d--〝,x[0] ;  
           return；  
｝else if x[k]<n&&k<n-1）｛/\*设置当期顶点的访问标志，继续下一个顶点\*/  
         （  ）  
          k=k+1;  
｝else｛/\*没有未被访问过的邻接顶点，回退到上一个顶点\*/  
         x[k]=0；  
         visited x[k]=0；  
         （  ）；  
    ｝  
｝  
｝  
【问题1】（10分）  
根据题干说明。填充C代码中的空（1）~（5）.  
【问题2】（5分）  
根据题干说明和C代码，算法采用的设计策略为（6），该方法在遍历图的顶点时，采用的是（7）方法（深度优先或广度优先）。

**试题分析**

**哈密顿图**是一个无向图，由天文学家哈密顿提出，由指定的起点前往指定的终点，途中经过所有其他节点且只经过一次。在图论中是指含有哈密顿回路的图，闭合的哈密顿路径称作哈密顿回路，含有图中所有顶点的路径称作哈密顿路径。  
**回溯法**是一种选优搜索法，又称为试探法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法，而满足回溯条件的某个状态的点称为“回溯点”。在包含问题的所有解的解空间树中，按照深度优先搜索的策略，从根结点出发深度探索解空间树。当探索到某一结点时，要先判断该结点是否包含问题的解，如果包含，就从该结点出发继续探索下去，如果该结点不包含问题的解，则逐层向其祖先结点回溯（其实回溯法就是对隐式图的深度优先搜索算法）。 若用回溯法求问题的所有解时，要回溯到根，且根结点的所有可行的子树都要已被搜索遍才结束。 而若使用回溯法求任一个解时，只要搜索到问题的一个解就可以结束。  
算法题历来都被认为是比较难的题，一个程序开发人员都不喜欢看别人的代码。但是要得分也不是太难。  
问题2比较容易得分，而且第二空就是个二选一的填空。只要了解到回溯法的相关原理，基本可以得满分。对于问题1就需要花一些心思，去读懂题干和代码，但是这里的第1空和第5空也是比较容易发挖出来的空。第一空是初始化第一个结点，第五空是此路不通，得回走，所以得退回。。

**试题答案**

（4）问题1：  
1、visited[0] = 1  
2、visited[x[k]] == 0  
3、visited[x[k]] == 1  
4、visited[x[k]] = 1  
5、k = k - 1  
问题2：  
6、回溯法  
7、深度优先

# **试题2(2017年上半年试题4)**

阅读下列说明和C代码，回答问题 1 至问题 3，将解答写在答题纸的对应栏内。  
【说明】  
假币问题：有n枚硬币，其中有一枚是假币，己知假币的重量较轻。现只有一个天平，要求用尽量少的比较次数找出这枚假币。  
【分析问题】  
将n枚硬币分成相等的两部分:  
(1)当n为偶数时，将前后两部分，即 1...n/2和n/2+1...0，放在天平的两端，较轻的一端里有假币，继续在较轻的这部分硬币中用同样的方法找出假币:  
(2)当n为奇数时，将前后两部分，即1..(n -1)/2和(n+1)/2+1...0，放在天平的两端，较轻的一端里有假币，继续在较轻的这部分硬币中用同样的方法找出假币；若两端重量相等，则中间的硬币，即第 (n+1)/2枚硬币是假币。  
【C代码】  
下面是算法的C语言实现，其中:  
coins[]： 硬币数组  
first，last：当前考虑的硬币数组中的第一个和最后一个下标  
   
#include <stdio.h>  
   
int getCounterfeitCoin(int coins[]， int first，int last)  
{  
      int firstSum = 0，lastSum = 0;  
      int ì;  
      If(first==last-1){        /\*只剩两枚硬币\*/  
            if(coins[first] < coins[last])  
                  return first;  
           return last;  
       }  
   
if((last - first + 1) % 2 ==0){        /\*偶数枚硬币\*/  
       for(i = first;i <(   1   );i++){  
             firstSum+= coins[i];  
        }  
        for(i=first + (last-first) / 2 + 1;i < last +1;i++){  
            lastSum += coins[i];  
        }  
        if(    2    ){  
            Return getCounterfeitCoin(coins,first,first+(last-first)/2;)  
        }else{  
            Return getCounterfeitCoin(coins,first+(last-first)/2+1,last;)  
        }  
}  
else{       /\*奇数枚硬币\*/  
        For(i=first;i<first+(last-first)/2;i++){  
               firstSum+=coins[i];  
        }  
        For(i=first+(last-first)/2+1;i<last+1;i++){  
               lastSum+=coins[i];  
        }  
        If(firstSum<lastSum){  
               return getCounterfeitCoin(coins,first,first+(last-first)/2-1);  
        }else if(firstSum>lastSum){  
               return getCounterfeitCoin(coins,first+(last-first)/2-1,last);  
        }else{  
            Return(   3    )  
        }  
     }  
}  
【问题一】  
根据题干说明，填充C代码中的空（1）-（3）  
【问题二】  
根据题干说明和C代码，算法采用了（   ）设计策略。  
函数getCounterfeitCoin的时间复杂度为（   ）（用O表示）。  
【问题三】  
若输入的硬币数为30，则最少的比较次数为（  ），最多的比较次数为（   ）。

**试题分析**

若输入30个硬币，找假硬币的比较过程为：  
         第1次：15 比 15，此时能发现假币在15个的范围内。  
         第2次：7  比  7，此时，如果天平两端重量相同，则中间的硬币为假币，此时可找到假币，这是最理想的状态。  
第3次：3  比 3，此时若平衡，则能找出假币，不平衡，则能确定假币为3个中的1个。  
         第4次：1  比 1，到这一步无论是否平衡都能找出假币，此时为最多比较次数。  
参考答案

**试题答案**

（4）问题1  
（1）first+(last-first)/2 或(first+last)/2                   
（2）firstSum<lastSum  
（3）first+(last-first)/2 或(first+last)/2  
问题2  
（4）分治法  
（5）O（nlogn）  
问题3  
（6）2     （7）4

# **试题3(2016年下半年试题4)**

阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。  
【说明】  
    模式匹配是指给定主串t和子串s，在主串t中寻找子串s的过程，其中s称为模式。如果匹配成功，返回s在t中的位置，否则返回-1 。  
   KMP算法用next数组对匹配过程进行了优化。KMP算法的伪代码描述如下：  
   1．在串t和串s中，分别设比较的起始下标i=j=0。  
   2．如果串t和串s都还有字符，则循环执行下列操作：  
    （1）如果j=-l或者t[i]=s[j]，则将i和j分别加1，继续比较t和s的下一个字符；  
    （2）否则，将j向右滑动到next[j]的位置，即j =next[j]。  
   3．如果s中所有字符均已比较完毕，则返回匹配的起始位置（从1开始）；否则返回-1．  
    其中，next数组根据子串s求解。求解next数组的代码已由get\_next函数给出。  
【C代码】  
（1）常量和变量说明  
    t，s：长度为悯铂Is的字符串  
    next:next数组，长度为Is  
（2）C程序  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
/\*求next[]的值\*/  
void get\_next( int \*next, char \*s, int Is)  {  
    int i=0，j=-1;  
    next[0]=-1;/\*初始化next[0]\*/  
    while(i < ls){/\*还有字符\*/  
    if(j==-1l ls[i]==s[j]){/\*匹配\*/  
        j++;  
        i++;  
    if( s[i]==s[j])  
       next[i] = next[j];  
    else  
       Next[i] = j;  
   }  
else  
  j = next[j];  
 }  
}  
 int kmp( int \*next, char \*t ,char \*s, int lt, int Is )  
 {  
       Int i= 0,j =0 ;  
       while (i < lt &&  （1）   ){  
          if( j==-1 ||     （2）  ){  
                 i ++ ;  
                 j ++ ;  
           } else  
                      （3）    ;  
}  
if (j >= ls)  
return     （4）   ;  
else  
    return -1;  
}  
【问题1】（8分）  
 根据题干说明，填充C代码中的空（1）～（4）.  
【问题2】（2分）  
根据题干说明和C代码，分析出kmp算法的时间复杂度为（5）（主串和子串的长度分别为It和Is，用O符号表示）。  
【问题3】（5分）  
根据C代码，字符串“BBABBCAC”的next数组元素值为（6）（直接写素值，之间用逗号隔开）。若主串为“AABBCBBABBCACCD”，子串为“BBABBCAC”，则函数Kmp的返回值是（7）。

**试题分析**

本题问题1根据KMP算法的伪代码描述进行推导。  
根据伪代码中第2步可以推导（1）是判断字符串s是否还有字符，即j<ls。i表示字符串t的下标，j表示字符串s的下标。  
根据伪代码第2.1步可以推导（2）是判断字符串t和字符串s当前位置的字符是否相同，即t[i]==s[j]。  
根据伪代码第2.2步可以推导（3）是当第2.1步判断条件不满足时，改变j所指向的字符位置。即调用函数get\_next(next, s, ls)，且j=next[j]。  
根据伪代码第3步可以推导（4）是返回匹配的起始位置。由于当前i所指向字符串中匹配子串的最后一个字符的位置，且已知子串的长度为ls。（4）的代码为i+1-ls。  
本题问题2是计算KMP算法的复杂度。算法的复杂度一般考虑最坏情况，那么在子串读到ls及主串读到It的时候是最坏情况。所以复杂度是O(It+Is)  
本题问题3中已知字符串“BBABBCAC”，则根据get\_next()函数可以求得next数组的元素值为[-1,-1,1,-1,-1,2,0,0]。并计算得到起始位置为6。

**试题答案**

（4）问题1：  
（1）：j<ls;  
（2）：t[i]==s[j];  
（3）：get\_next(next, s, ls);  
               j=next[j];  
（4）：i+1-ls;  
问题2：O(It+Is)  
问题3：（6）：[-1,-1,1,-1,-1,2,0,0]，（7）6。

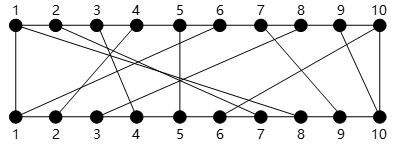
# **试题4(2016年上半年试题4)**

试题四（共15分）

    阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。

【说明】

    在一块电路板的上下两端分别有n个接线柱。根据电路设计，用(i,π(i))表示将上端接线柱i与下端接线柱π(i)相连，称其为该电路板上的第i条连线。如图4-1所示的π(i)排列为{8,7,4,2,5,1,9,3,10,6}。对于任何1≤i<j≤n，第i条连线和第j条连线相交的充要条件是π(i)>π(j)。



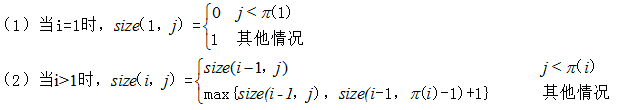
**图4-1 电路布线示意**

    在制作电路板时，要求将这n条连线分布到若干绝缘层上，在同一层上的连线不相交。现在要确定将哪些连线安排在一层上，使得该层上有尽可能多的连线，即确定连线集Nets={(i,π(i))，1≤i≤n}的最大不相交子集。

【分析问题】

    记N(i,j)={t|(t,π(t))∈Nets,t≤i,π(t)≤j}。N(i,j)的最大不相交子集为MNS(i,j)，size(i,j)=|MNS(i,j)|。

    经分析，该问题具有最优子结构性质。对规模为n的电路布线问题，可以构造如下递归式：



【C代码】

    下面是算法的C语言实现。

（1）变量说明

    size[i][j]：上下端分别有i个和j个接线柱的电路板的第一层最大不相交连接数

    pi[i]： π(i)，下标从1开始

（2）C程序

#include "stdlib.h"

#include <stdio.h>

#define  N  10    /\*问题规模\*/

int m=0；    /\*牢记录最大连接集合中的接线柱\*/

Void maxNum(int pi[],int size[N+1][N+1],int n)  {/\*求最大不相交连接数\*/

    int i, j;

    for(j=0; j < pi[1]; j++)   size[1][j] = 0;   /\*当j<π(1)时  \*/

    for(j=pi[1];j<=n;j++)    （1）   ; /\*当j>=π(1)时  \*/

    for(i=2; i < n; i++)   {

        for(j=0; j < pi[i]; j++)    （2）   ; /\*当j<pi[i]时  \*/

        for(j=pi[i];j<=n；j++)  {/\*当j>=c[i]时,考虑两种情况\*/

           size[i][j]=size[i-1][j]>=size[i-1][pi[i]-1]+1 ?size[i-1][j]:size[i-1][pi[i]-1]+1;

        }

    }

    /\*最大连接数  \*/

    size[n][n]=size[n-1][n]>=size[n-1][pi[n]-1]+1 ? size[n-1][n]:size[n-1][pi[n]-1]+1;

}

/\*构造最大不相交连接集合，net[i]表示最大不相交子集中第i条连线的上端接线柱的序号  \*/

void constructSet（int pi[],int size[N+1][N+1],int n,int net[n]）{

    int i,j=n;

    m=0；

    for(i=n ; i>1 ; i--)    {    /\*从后往前\*/

        if(size[i][j]!=size[i-1][j]){  /\*(i,pi[i])是最大不相交子集的一条连线\*/

              （3）  ;    /\*将i记录到数组net中，连接线数自增1\*/

            j= pi[i]-1;    /\*更新扩展连线柱区间\*/

        }

    }

    if(j>=pi[1])  net[m++]=1;  /\*当i=1时\*/

}

【问题1】（6分）

    根据以上说明和C代码，填充C代码中的空（1）～（3）。

【问题2】（6分）

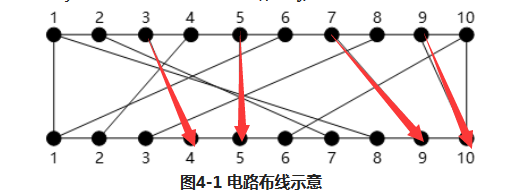
    据题干说明和以上C代码，算法采用了  （4）  算法设计策略。

    函数maxNum和constructSet的时间复杂度分别为  （5）   和   （6）  （用O表示）。

【问题3】（3分）

    若连接排列为{8,7,4,2,5,1,9,3,10,6}，即如图4-1所示，则最大不相交连接数为   （7）   ，包含的连线为  （8）   （用(i,π(i))的形式给出）。

**试题分析**

这个是动态规划问题，不想交的平行线，算法思路来才能完整。  
设a[i][j]为上端接线柱i与下端接线柱j前的最大不相交子集，则：  
若i与j不相连，则i与j前的最大不想交子集等于i与j - 1前或i - 1与j前的最大不相交子集的最大值，即a[i][j] = max(a[i][j - 1], a[i - 1][j])  
若i与j相连，则i与j前的最大不想交子集等于i - 1与j - 1前的最大不想交子集加1，即a[i][j] = a[i - 1][j - 1] + 1  
题目的意思就是要求出，没有交叉的这种连线的数量达到最大的情况。此时，有4条这样的线不会交叉，所以是大不相交子集连接数为4。如果你能找到5条这样不交叉的线，则是5。就这个意思。  


**试题答案**

（4）

【问题1】

（1）size[i][j]=1

（2）size[i][j]=size[i-1][j]

（3）net[m++]=i;

【问题2】

（4）动态规划算法

（5）O(n2)

（6）O(n)

【问题3】

（7）4  
（8）（3，π（3），（5，π（5）），（7，π（7）），（9，π（9））  
或：（3，4），（5，5），（7，9），（9，10）

# **试题5(2015年下半年试题4)**

    阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。  
【说明】  
    计算两个字符串x和y的最长公共子串（Longest Common Substring）。  
    假设字符串x和字符串y的长度分别为m和n，用数组c的元素c[i][j]记录x中前i个字符和y中前j个字符的最长公共子串的长度。  
    c[i][j]满足最优子结构，其递归定义为：  
  
 IMG_270  
    计算所有c[i][j](0 ≤i ≤ m，0 ≤j ≤ n)的值，值最大的c[i][j]即为字符串x和y的最长公共子串的长度。根据该长度即i和j，确定一个最长公共子串。  
【C代码】  
(1)常量和变量说明  
    x，y：长度分别为m和n的字符串  
    c[i][j]：记录x中前i个字符和y中前j个字符的最长公共子串的长度  
    max：x和y的最长公共子串的长度  
    maxi, maXj：分别表示x和y的某个最长公共子串的最后一个字符在x和y中的位置（序号）  
  (2)C程序  
  
#include <stdio.h>  
#include <string.h>  
  
int c[50][50];  
int maxi;  
int maxj;  
  
int lcs(char \*x, int m, char \*y, int n)     {  
     int i, j;  
     int max= 0;  
     maxi= 0;  
     maxj = 0;  
  
  
     for ( i=0; i<=m ; i++)            c[i][0] = 0;  
     for (i =1; i<= n; i++)              c[0][i]=0;  
     for (i =1; i<= m; i++)    {  
         for (j=1; j<= n; j++)    {  
            if (    (1)    )    {  
             c[i][j] = c[i -1][j -1] + 1;  
             if(max<c[i][j]) {  
                 (2)   ;  
               maxi = i;  
               maxj =j;  
             }  
        }  
        else     (3)    ;  
       }  
    }  
     return max;  
}     
void printLCS(int max, char \*x) {  
          int i= 0;  
       if (max == 0)        return;  
      for (  (4)    ; i < maxi; i++)  
             printf("%c",x[i]);  
}  
void main(){  
char\* x= "ABCADAB";  
char\*y= "BDCABA";  
 int max= 0;  
 int m = strlen(x);  
 int n = strlen(y);  
  
 max=lcs(x,m,y,n);  
 printLCS(max , x);  
}  
【问题1】（8分）     
    根据以上说明和C代码，填充C代码中的空（1）～（4）。  
【问题2】（4分）     
    根据题干说明和以上C代码，算法采用了 （5） 设计策略。  
    分析时间复杂度为 （6） （用O符号表示）。  
【问题3】（3分）  
    根据题干说明和以上C代码，输入字符串x= "ABCADAB’，'y="BDCABA",则输出为 （7） 。

**试题分析**

首先对于C语言算法题，一般的解题思路是先解决除程序填空以外的问题，这些问题弄清楚，有利于程序填空部分的分析。

第一步，分析程序所采用的算法，常见的算法包括：分治法、动态规划法、回溯法、贪心法。本题中要求的是两个串的最长公共子串，在程序中采用了数组来记录子问题的中间结果，这一特征与动态规划法的做法非常吻合，所以应选动态规划法。

第二步，解决“输入字符串x= "ABCADAB’，'y="BDCABA",则输出为 （7） ”的问题，该问题相对容易解决，因为题目已告知程序的作用是求最长公共子串，而且从程序的输出函数可以看出，要输出的，只有子串，没有其它信息，所以我们只需要手动求两个串的公共子串，并写出答案即可。两个串第一个公共子串明显是“AB”，所以输出的结果为“AB”。

第三步，求时间复杂度，由于程序中最多双重循环，其中外层循环的规模为m，而内层循环的规模为n，所以时间复杂度为O(m\*n)。

第四步，也是最难的一步，是解决程序填空的问题。动态规划的问题，一般都会给出递归定义式，这个式子，往往是多个空的关键点。以本题为例，前3空均与式子相关，式子中明确说明了x[i-1]=x[j-1]时c[i][j]的值是多少，而其它情况c[i][j]均等于0，此时可以了解到前3空的答案。而第4空是用于打印结果，由于maxi记录了子串末尾+1的位置信息，子串长度为max，所以用maxi-max定位至子串开始位置，以便打印子串。

**试题答案**

（4）【问题1】  
（1）x[i-1] == y[j-1]  
（2）max=c[i][j]  
（3）c[i][j]=0  
（4）i=maxi-max  
【问题2】  
（5）动态规划法                          
（6）O(m\*n)  
【问题3】  
（7）AB

# **试题6(2015年上半年试题4)**

    阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。

【说明】

    n-皇后问题是在n行n列的棋盘上放置n个皇后，使得皇后彼此之间不受攻击，其规则是任意两个皇后不在同一行、同一列和相同的对角线上。

    拟采用以下思路解决n-皇后问题：第i个皇后放在第i行。从第一个皇后开始，对每个皇后，从其对应行（第i个皇后对应第i行）的第一列开始尝试放置，若可以放置，确定该位置，考虑下一个皇后；若与之前的皇后冲突，则考虑下一列；若超出最后一列，则重新确定上一个皇后的位置。重复该过程，直到找到所有的放置方案。

【C代码】

    下面是算法的C语言实现。

(1)常量和变量说明

    pos：一维数组，pos[i]表示第i个皇后放置在第i行的具体位置

    count：统计放置方案数

    i，j，k：变量

    N：皇后数

(2)C程序

#include <stdio.h>

#include <math.h>

#define N4

/\*判断第k个皇后目前放置位置是否与前面的皇后冲突\*/

in isplace(int pos[], int k) {

    int i;

        for(i=1; i<k; i++) {

          if(  （1）  || fabs(i-k)  ══ fabs(pos[i] - pos[k])) {

            return 0;

          }

        }

        return 1;

}

int main() {

    int i,j,count=1;

    int pos[N+1];

    //初始化位置

    for(i=1; i<=N; i++) {

        pos[i]=0;

        }

           （2）    ；

        while(j>=1) {

            pos[j]= pos[j]+1；

             /\*尝试摆放第i个皇后\*/

            while(pos[j]<=N&&    （3）\_) {

                pos[j]= pos[j]+1;

            }

            /\*得到一个摆放方案\*/

            if(pos[j]<=N&&j══ N) {

                printf("方案%d: ",count++);

                for(i=1; i<=N; i++){

                    printf("%d  ",pos[i]);

                }

                printf("\n");

          }

          /\*考虑下一个皇后\*/

          if(pos[j]<=N&&  （4）  ) {

              j=j+1;

          } else{ //返回考虑上一个皇后

              pos[j]=0;

                 （5）    ;

          }

    }

    return 1;

}

【问题1】（10分）

    根据以上说明和C代码，填充C代码中的空（1）～（5）。

【问题2】（2分）

    根据以上说明和C代码，算法采用了    （6）   设计策略。

【问题3】（3分）

    上述C代码的输出为：

       （7）   。

**试题分析**

本题考查算法设计和 C 程序设计语言的相关知识。  
此类题目要求考生认真阅读题目，理解算法思想，并思考将算法思想转化为具体的程序设计语言的代码。  
【问题1】  
根据题干描述。空（1）所在的代码行判断皇后合法放置的约束条件，即不在同一行，这通过把第 i 个皇后放在第 i 行实现，条件 "fabs(i-k) = fabs(pos[i]-pos[k])" 判断的是当前摆放的皇后是否与之前摆放的皇后在同一对角线上。因此，空（1）判断的是当前摆放的皇后是否和之前摆放的皇后在同一列上，即应填入"pos[i]=pos[k]" 。  
根据算法思想和主函数上下文，空（2）处应该考虑第 1 个皇后，即初始化 j为1，空（2）填写"j=1"。空（3）所在的行是判断放置第j 个皇后的位置是否合适，"pos[j]<= N"   表示在该行的合法列上，但还需要进一步判断是否与前面的皇后有冲突，根据满足条件后的语句，尝试放入下一列，因此空（3）处填入"!isplace(pos，j) 。根据前面的注释，空（4）所在的行是考虑下一个皇后，其条件是，当前皇后找到了合适的位置，而且还存在下一个皇后，因此空（4）处应填入"j<N" 。根据下面的注释，若当前皇后没 有找到合适的位置，则应回溯，即再次考虑上一个皇后的位置，因此空（5）处填入 "j=j-1"。  
【问题 2 】  
从上述题干的叙述和 C 代码很容易看出，从第一个皇后开始，对每个皇后总是从第一个位置开始尝试，找到可以放置的合法位置；若某个皇后在对应的行上没有合法位置，则回溯到上一个皇后，尝试将上一个皇后放置另外的位置。这是典型的深度优先的系统搜索方式，即回溯法的思想。  
【问题3】  
四皇后问题的答案为：  
方案 1：       2    4    1    3  
方案 2：       3    1    4    2  
如表 4-1 所示：

**表4-1**

方案1

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

方案2

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**试题答案**

（4）

【问题1】

（1）pos[i] ==pos[k]

（2）j=1

（3）isplace(pos,j)==0

（4）j<N

（5）j=j-1

【问题2】

（6）回溯法

【问题3】

（7）  
方案1：2 4 1 3

方案2：3 1 4 2

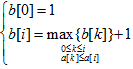
# **试题7(2014年下半年试题4)**

    阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。

【说明】

    计算一个整数数组a的最长递增子序列长度的方法描述如下：

    假设数组a的长度为n，用数组b的元素b[i]记录以a[i](0≤i<n)为结尾元素的最长递增予序列的长度，则数组a的最长递增子序列的长度为IMG_271；其中b[i]满足最优子结构，可递归定义为：



【C代码】

    下面是算法的C语言实现。

    （1）常量和变量说明

        a：长度为n的整数数组，待求其最长递增子序列

        b：长度为n的数组，b[i]记录以a[i](0≤i<n)为结尾元素的最长递增子序列的长

度，其中0≤i<n

        len：最长递增子序列的长度

        i,j：循环变量

        temp：临时变量

    （2）C程序

#include <stdio.h>

int maxL(int\*b, int n) {

  int i, temp=0;

  for(i=0; i<n; i++) {

    if(b[i]>temp)

      temp=b[i];

  }

  return temp;

}

int main() {

  int n, a[100], b[100], i, j, len;

  scanf("%d", &n);

  for(i=0; i<n; i++) {

    scanf("%d", &a[i]);

  }

      (1)    ;

  for(i=1; i<n; i++) {

    for(j=0, len=0;     (2)    ; j++) {

      if(    (3)     && len<b[j])

        len=b[j];

    }

        (4)    ;

  }

  Printf("len:%d\n", maxL(b,n));

  printf("\n");

}

【问题1】（8分）

    根据说明和C代码，填充C代码中的空（1）～（4）。

【问题2】（4分）

    根据说明和C代码，算法采用了 （5） 设计策略，时间复杂度为 （6） （用O符号表示）。

【问题3】（3分）

    已知数组a={3,10,5,15,6,8}，根据说明和C代码，给出数组b的元素值。

**试题分析**

本题考察算法设计与分析技术以及算法的C语言实现，是比较传统的题目，要求考生细心分析题目中所描述的内容。

（1） 根据题中说明，b数组记录最长递增子序列的长，故应初始化b[0]=1，这是第一问的答案。两重for循环中，第一重是从a数组的第二个元素开始，考虑每个子数组a[0...i]的最长递增子序列的长度，第二重是具体的计算过程。考虑子数组a[0..i]，其最长递增子序列的长度应该等于子数组a[0..i-1]中的比元素a[i]小的元素的最长递增子序列的长度加1，当然，可能存在多个元素比元素a[i]小，那么存在多个最长递增子序列的长度，此时，取最大者。因此，空（2）填写“j<i”,即考虑子数组a[0..i-1]第三问为a[j]<=a[i]，第四问为b[i]=len+1。

（2）算法将待求解问题分解成若干个子问题，先求解子问题，然后从这些子问题的解得到原问题的解。使用的是动态规划的思想。时间复杂度计算最坏情况下的运算次数，最坏情况时i和j都从1跑到n,故运算n的平方次。算法的时间复杂度为O(n2)。

（3）初始b[0]=1，a[0]=3，a[1]=10进入时b[1]=2，a[2]=5进入时有3、5的序列故b[2]=2，a[3]=15进入时有3、10、15，故子序列为3，a[4]=6时有子序列3、5、6，故为3，当最后一个元素8进入时有3、5、6、8，故b[5]=4。所以b=[1,2,2,3,3,4]。

**试题答案**

（4）

【问题1】

（1）b[0]=1

（2）j<i

（3）a[j]<=a[i]

（4）b[i]=len+1

【问题2】

（5）动态规划法   
（6）O(n2)

【问题3】

b={1,2,2,3,3,4}

# **试题8(2014年上半年试题4)**

    阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。  
【说明】  
    采用归并排序对n个元素进行递增排序时，首先将n个元素的数组分成各含n/2个元素的两个子数组，然后用归并排序对两个子数组进行递归排序，最后合并两个已经排好序的子数组得到排序结果。  
    下面的C代码是对上述归并算法的实现，其中的常量和变量说明如下：  
    arr：待排序数组  
    p,q,r：一个子数组的位置从p到q，另一个子数组的位置从q+1到r  
    begin,end：待排序数组的起止位置  
    left,right：临时存放待合并的两个子数组  
    n1,n2：两个子数组的长度  
    i,j,k：循环变量  
    mid：临时变量  
【C代码】

#inciude<stdio.h>

#inciude<stdlib.h>

#define MAX 65536

void merge(int arr[],int p,int q,int r) {

    int \*left, \*right;

    int n1,n2,i,j,k;

    n1=q-p+1;

    n2=r-q;

    if((left=(int\*)malloc((n1+1)\*sizeof(int)))=NULL) {

        perror("malloc error");

        exit(1);

    }

    if((right=(int\*)malloc((n2+1)\*sizeof(int)))=NULL) {

        perror("malloc error");

        exit(1);

    }

    for(i=0;i<n1;i++){

        left[i]=arr[p+i]；

    }

    left[i]=MAX；

    for(i=0; i<n2; i++){

        right[i]=arr[q+i+1]

    }

    right[i]=MAX;

    i=0; j=0;

    for(k=p;   （1）  ; k++) {

        if(left[i]> right[j]) {

              （2）  ;

            j++;

        }else {

            arr[k]=left[i]；

            i++；

        }

    }

}

void mergeSort(int arr[],int begin,int end){

    int mid;

    if(  （3）  ){

        mid=(begin+end)/2;

        mergeSort(arr,begin,mid);

          （4）  ;

        merge(arr,begin,mid,end);

    }

}

【问题1】

    根据以上说明和C代码，填充1-4。

【问题2】

    根据题干说明和以上C代码，算法采用了（5）算法设计策略。

    分析时间复杂度时，列出其递归式位（6），解出渐进时间复杂度为（7）（用O符号表示）。空间复杂度为（8）（用O符号表示）。

【问题3】

    两个长度分别为n1和n2的已经排好序的子数组进行归并，根据上述C代码，则元素之间比较次数为（9）。

**试题分析**

根据题目中的参数说明，void  merge(int arr[],int p,int q,int r)是将数组arr[p...q]和数组arr[q+1...r]进行合并成一个排序的数组，因此合并之后数组的长度为r-q+1，也就是k<=r；数组arr存入子数组arr[p...q]、arr[q+1...r]当前进行比较的最小值，因此当left[i]> right[j]时，数组arr中存入right[i]，即arr[k]=right[j]；  
void mergeSort(int arr[],int begin,int end)是指将数组arr递归进行划分，直到分成多个由一个元素组成的子数组时，停止划分，此时也就是begin==end，因此（3）处为begin<end，也就是只要begin!=end则继续划分。划分的时候每次分成两半，两半再分别递归，因此mergeSort(arr,begin,mid);mergeSort(arr,mid+1,end);；  
很明显归并排序使用的分治算法，每次讲数组分割成两个小的子数组。  
假设对n个元素的数组进行归并排序时间复杂度为T(n)，则分成两个小的子数组后分别进行排序所需的时间为T(n/2)，两个子数组则时间复杂度为2T(n/2)，再加上归并的时间O(n)，即可得出答案。

**试题答案**

（4）

【问题1】（8分）  
（1）k<=r  
（2）arr[k]=right[j]  
（3）begin<end  
（4）mergeSort(arr,mid+1,end)

【问题2】（5分）  
（5）分治  
（6）T(n)=2T(n/2)+O(n)  
（7）O(nlogn)  
（8）O(n)

【问题3】（2分）  
（9）n1+n2

# **试题9(2013年下半年试题4-7)**

    阅读下列说明和C代码，回答问题1至问题3，将解答填入答题纸的对应栏内。

【说明】

    某工程计算中要完成多个矩阵相乘（链乘）的计算任务。

    两个矩阵相乘要求第一个矩阵的列数等于第二个矩阵的行数，计算量主要由进行乘法运算的次数决定。采用标准的矩阵相乘算法，计算Am×n\*Bn×p，需要m\*n\*p次乘法运算。

    矩阵相乘满足结合律，多个矩阵相乘，不同的计算顺序会产生不同的计算量。以矩阵A110×100，A2100×5，A35×50三个矩阵相乘为例，若按（A1\*A2）\*A3计算，则需要进行10\*100\*5+10\*5\*50=7500次乘法运算；若按A1\*（A2\*A3）计算，则需要进行100\*5\*50+10\*100\*50=75000次乘法运算。可见不同的计算顺序对计算量有很大的影响。

    矩阵链乘问题可描述为：给定n个矩阵<A1,A2,….An>，矩阵Ai的维数为pi-1×pi，其中i = 1,2,….n。确定一种乘法顺序，使得这n个矩阵相乘时进行乘法的运算次数最少。

    由于可能的计算顺序数量非常庞大，对较大的n，用蛮力法确定计算顺序是不实际的。经过对问题进行分析，发现矩阵链乘问题具有最优子结构，即若A1\*A2\*…\*An的一个最优计算顺序从第k个矩阵处断开，即分为A1\*A2\*….Ak和Ak+1\*Ak+2\*…\*An两个子问题，则该最优解应该包含A1\*A2\*…\*Ak的一个最优计算顺序和Ak+1\*Ak+2\*…An的一个最优计算顺序。据此构造递归式，

 点击查看大图

    其中，cost[i][j]表示Ai+1\*Ai+2\*...Aj+1的最优计算的计算代价。最终需要求解cost[0][n-1]。

【C代码】

    算法实现采用自底向上的计算过程。首先计算两个矩阵相乘的计算量，然后依次计算3个矩阵、4个矩阵、…、n个矩阵相乘的最小计算量及最优计算顺序。下面是算法的C语言实现。

（1）主要变量说明

n：矩阵数

seq[]：矩阵维数序列

cost[][]：二维数组，长度为n\*n，其中元素cost[i][j]表示Ai+1\*Ai+2\*…Aj+1的最优计算的计算代价

trace[][]：二维数组，长度为n\*n，其中元素trace[i][j]表示Ai+1\*Ai+2\*Aj+1的最优计算对应的划分位置，即k

（2）函数cmm

#define  N  100

int cost[N][N];

int trace[N][N];

int cmm(int n,int seq[]){

    int tempCost;

    int tempTrace;

    int i,j,k,p;

    int temp;

    for( i=0;i<n;i++){ cost[i][i] = 0;}

    for(p=1;p<n;p++){

        for(i=0;  （1） ;i++){

              （2）  ;

            tempCost = -1;

            for(k = i;k<j;k++){

            temp=  （3）  ;

                if(tempCost==-1||tempCost>temp){

                    tempCost = temp;

                      （4）  ;

                }

            }

            cost[i][j] = tempCost;

            trace[i][j] = tempTrace;

        }

    }

    return cost[0][n-1];

}

【问题1】（8分）  
    根据以上说明和C代码，填充C代码中的空（1）~（4）。  
【问题2】（4分）  
    根据以上说明和C代码，该问题采用了 （5） 算法设计策略，时间复杂度 （6） 。（用O符号表示）  
【问题3】（3分）  
    考虑实例n=6，各个矩阵的维数：A1为5\*10，A2为10\*3，A3为3\*12，A4为12\*5，A5为5\*50，A6为50\*6，即维数序列为5,10,3,12,5,50,6。则根据上述C代码得到的一个最优计算顺序为（7） （用加括号方式表示计算顺序），所需要的乘法运算次数为 （8） 。

**试题分析**

在解答本题时，需要注意的第一个问题便是矩阵的乘法到底是怎么进行的。

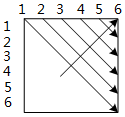
       一个n行m列的矩阵可以乘以一个m行p列的矩阵，得到的结果是一个n行p列的矩阵，其中的第i行第j列位置上的数等于前一个矩阵第i行上的m个数与后一个矩阵第j列上的m个数对应相乘后所有m个乘积的和。如：

IMG_274

    在本题中，题干部分提到“发现矩阵链乘问题具有最优子结构”，这是利用动态规划法求解最优解问题的典型特征。所以（5）应填动态规划法。

    接下来分析（1）-（4）空，这几个空中，最容易回答的是（3）和（4）。（3）空可通过题目给出的递归式分析得到，其中cost数组部分与公式完全一致，而p数组在程序中是seq，所以回答时修正即可，（3）填：cost[i][k]+cost[k+1][j]+seq[i]\*seq[k+1]\*seq[j+1]。第（4）空的上一句为：tempCost = temp，即保存当前状态最优解，由于在保存最优解时，不仅涉及cost的记录，还涉及其位置k的记录，所以需要在此进行tempTrace=k的操作。

       （1）与（2）相对复杂，其中（1）是对i值范围的确定，而（2）是对j的赋值操作（由于后面用到了j，但程序中没有对j的赋值，从而断定该空是对j的赋值）。两者一并起到一个效果，对cost数组操作时的操作范围与顺序。由于在进行矩阵链乘操作时，分析解空间所用到的是cost右上角的三角矩阵，而操作时，是对这个三角矩阵从左至右，呈斜线的访问（如图所示）。所以（1）和（2）分别填i<n-p和j=i+p。



    该程序由于涉及3重循环，所以时间复杂度为：O（n3）。通过手动运行程序的方式可知最优解为：

       （A1A2）（（A3A4）（A5A6））。

       总计算次数为2010。

**试题答案**

（4）

【问题1】

（1）i<n-p

（2）j=i+p

（3）cost[i][k]+cost[k+1][j]+seq[i]\*seq[k+1]\*seq[j+1]

（4）tempTrace=k

【问题2】

（5）动态规划法

（6）O（n3）

【问题3】

（7）((A1A2)((A3A4)(A5A6)))

（8）2010

# **试题10(2013年上半年试题4-6)**

设有m台完全相同的机器运行n个独立的任务，运行任务i所需的时间为ti，要求确定一个调度方案，使得完成所有任务所需要的时间最短。  
假设任务已经按照其运行时间从大到小排序，算法基于最长运行时间作业优先的策略，按顺序先把每个任务分配到一台机器上，然后将剩余的任务一次放入最先空闲的机器。  
【C代码】  
下面是算法的C语言实现。  
1.常量和变量说明  
m：机器数  
n：任务数  
t[]：输入数组，长度为n，下标从0开始，其中每个元素表示任务的运行时间，下标从0开始。  
s[][]：二位数组，长度为m\*n，下标从0开始，其中元素s[i][j]表示机器i运行的任务j的编号。  
d[]：数组，长度为m其中元素d[i]表示机器i的运行时间，下标从0开始。  
count[]：数组，长度为m，下标从0开始，其中元素count[i]表示机器i运行的任务数。  
i：循环变量。  
j：循环变量。  
k：临时变量。  
max：完成所有任务的时间。  
min：临时变量。

2.函数schedule  
void  schedule(){  
int i,j,k,max=0;  
 for( i=0;i<m;i++){  
      d[i]=0;  
      for(j=0;j<n;j++){  
         s[i][j]=0;  
      }  
  }  
  for(i=0;i<m;i++){   //分配前m个任务  
     s[i][0]=i;  
      (1)     ;  
      count[i]=1;  
   }  
   for(  (2)  ;i<n;i++){             //分配后n~m个任务  
     int min = d[0];  
     k=0;  
     for(j=1;j<m;j++){             //确定空闲时间  
        if(min>d[j]){  
          min = d[j];  
           k=j;                    //机器k空闲  
      }  
    }  
   (3)   ;  
   count[k] = count[k]+1;  
   d[k] = d[k]+t[i];  
}

   for(i =0;i<m;i++){               //确定完成所有任务所需要的时间  
      if(  (4)  ){  
         max=d[i];  
      }  
   }  
}

【问题1】（8分）  
根据说明和C代码，填充C代码中的空（1）~（4）。  
【问题2】（2分）  
根据说明和C代码，该问题采用了 （5）   算法设计策略，时间复杂度   （6）  （用O符号表示）  
【问题3】（5分）  
考虑实例m=3（编号0~2），n=7（编号0~6），各任务的运行时间为{16，14，6，5，4，3，2}。则在机器0、1和2上运行的任务分别为（7）、（8）和（9）（给出任务编号）。从任务开始运行到完成所需的时间为（10）。

**试题分析**

本题考查算法的设计和分析技术中的贪心算法。  
贪婪算法（Greedy algorithm）是一种对某些求最优解问题的更简单、更迅速的设计技术。用贪婪法设计算法的特点是一步一步地进行，常以当前情况为基础根据某个优化测度作最优选择，而不考虑各种可能的整体情况，它省去了为找最优解要穷尽所有可能而必须耗费的大量时间，它采用自顶向下，以迭代的方法做出相继的贪心选择，每做一次贪心选择就将所求问题简化为一个规模更小的子问题，通过每一步贪心选择，可得到问题的一个最优解，虽然每一步上都要保证能获得局部最优解，但由此产生的全局解有时不一定是最优的，所以贪婪法不要回溯。  
【问题1】  
根据上述思想和题中的说明，首先将s[][]和d[]数组初始化为0，然后要做的就是按要求“算法基于最长运行时间作业优先的策略，按顺序先把每个任务分配到一台机器上”，可以推断（1）处为d[i] = t[i]，此后需将剩下的n-m个任务按顺序分配给空闲的机器，故（2）处将i初始化为以m为起始的任务，即i=m，（3）处所在的位置是分配后n-m个任务，在这个过程中，必须要对s矩阵的内容进行修改，但目前已经出的代码没有这个内容，所以此处必然是对s的修改。从对s矩阵的注释可以了解到，s[i][j]表示机器i运行的任务j的编号，此时涉及任务的机器号为k，而待分配的任务i是机器的第count[k]个任务，即s[k][count[k]]=i，（4）处已经完成了任务的运行，此处需要统计所有机器所运行任务的最长时间，对于每个机器i的运行时间为d[i]，存在d[i]大于当前的最大时间Max，就将当前机器的运行时间d[i]赋给Max，即Max<d[i]。  
【问题2】  
根据以上分析，（5）处采用了贪心算法的策略，而时间复杂度由算法中的两个嵌套for循环和两个非嵌套for循环确定，即为O(mn)。  
【问题3】  
根据题中算法的思想将任务的前三个任务分给三个机器，再将接下来的任务分给最先空闲的机器，故可知机器0运行任务0，机器1运行任务1、5，机器3运行任务2、3、4、6；且运行的最长时间为17。

**试题答案**

（4）

【问题1】  
（1）d[i] = t[i] （2）i=m  （3）s[k][count[k]]=i  （4）max<d[i]  
【问题2】  
（5）贪心  （6）O(mn)  
【问题3】  
（7）0  （8）1、5   （9）2、3、4、6   （10）17

# **试题11(2012年下半年试题4)**

阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。  
【说明】  
设有n个货物要装入若干个容量为C的集装箱以便运输，这n个货物的体积分别为{S1，S2，．．．，Sn}，且有si≤C(1≤i≤ n)。为节省运输成本，用尽可能少的集装箱来装运这n个货物。  
下面分别采用最先适宜策略和最优适宜策略来求解该问题。  
最先适宜策略( firstfit)首先将所有的集装箱初始化为空，对于所有货物，按照所给的次序，每次将一个货物装入第一个能容纳它的集装箱中。  
最优适宜策略( bestfit)与最先适宜策略类似，不同的是，总是把货物装到能容纳它且目前剩余容量最小的集装箱，使得该箱子装入货物后闲置空间最小。  
【C代码】  
下面是这两个算法的C语言核心代码。  
(1)变量说明  
    n：货物数  
    C：集装箱容量  
    s：数组，长度为n，其中每个元素表示货物的体积，下标从0开始  
    b：数组，长度为n，b[i]表示第i+1个集装箱当前已经装入货物的体积，下标从0开始  
    i，j：循环变量  
    k：所需的集装箱数  
    min:当前所用的各集装箱装入了第i个货物后的最小剩余容量  
    m:当前所需要的集装箱数  
    temp：临时变量  
(2)函数firstfit  
int firstfit(){  
    inti，j；  
    k=0：  
    for(i=0；i<n；i++){  
  b[i]=0；  
    }  
    for（i=0；i<n；i++）{  
      (1)；  
      while(C-b[j]<s[i]){  
    j++;  
      }  
      (2)；  
k=k>(j+1)?k：(j+1)；  
  }  
  returnk；  
}  
(3)函数bestfit  
int bestfit() {  
  int i，j，min，m，temp;  
  k=0；  
  for（i=0；i<n；i++）{  
    b[i]=0；  
  }  
  for (i=0；i<n；i++）{  
    min=C；  
    m=k+l；  
    for(j=0;j< k+l；j++){  
  temp=C- b[j] - s[i];  
  if(temp>0&&temp< min){  
    (3)    ；  
    m=j，  
  }  
    }  
    (4)；  
    k=k>(m+1)?k:(m+1)；  
  }  
  return k；  
}

【问题1】（8分）  
根据【说明】和【C代码】，填充C代码中的空(1)～(4)。  
【问题2】（4分）  
 根据【说明】和【C代码】，该问题在最先适宜和最优适宜策略下分别采用了(5) 和(6)算法设计策略，时间复杂度分别为 (7) 和 (8)（用O符号表示）。  
【问题3】（3分）  
考虑实例n= 10，C= 10，各个货物的体积为{4，2，7，3，5，4，2，3，6，2}。该实例在最先适宜和最优适宜策略下所需的集装箱数分别为(9)和(10)。考虑一般的情况，这两种求解策略能否确保得到最优解？(11)  （能或否）

**试题分析**

本题考查最先适宜策略和最优适宜策略。这两种策略在题目的描述中给出了清楚的解析，对于最先适宜策略，其关键是每次将一个货物装入第一个能容纳它的集装箱中；而对于最优适宜策略，则总是把货物装到能容纳它且目前剩余容量最小的集装箱。  
下面我们来具体分析程序。函数firstfit()是实现最先适宜策略的，从程序不难看出，第（1）空所在的for循环，就是要将n各货物装入到集装箱。根据算法的描述，是依次从第一个集装箱找，找到合适的就装入货物，依次没装入一个货物，都是依次从第一个集装箱找。结合后面的程序不难知道j标识这当前是第几个集装箱。因此每装入一个货物后，要将j清0，标识从头再找，因此第（1）空的答案是j=0。而接下来的while循环，从其条件表达式C-b[j]<s[i]不难知道，是比较当前集装箱和当前货物的体积大小，如果当前集装箱体积小，则比较下一个集装箱，否则，就应该将货物装入该集装箱，并且调整集装箱剩余体积的大小，在本题中，这个是通过数组b来实现的，因此第（2）空的答案应该为b[j]=b[j]+s[i]。  
第（3）和第（4）空是在函数bestfit()下，这个函数是实现最优适宜策略的。从程序中不难看出，for(j=0;j< k+l；j++)就是要在众多的集装箱中找到最合适的集装箱，而第（3）空是条件if(temp>0&&temp< min)成立时，执行的语句，该条件成立，表示当前找到的集装箱比原来确定的集装箱更合适，而最合适的集装箱的剩余体积存放在min中，因此第3空的答案为min= temp，而循环结束后，就应该找到了合适的集装箱，这时应该将货物存放到集装箱里面，即第（4）空的答案为b[m]= b[m]+s[i]。  
在本题中，不管是采用最先适宜策略，还是最优适宜策略，他们都是根据不同策略选择目前看来最优的情况，这都属于贪心算法的思想。从两个函数不难看出，其时间复杂度是一样的，都是O(n2)。  
第3个问题，其实是这个题目中最简单的问题，也是算法的一个实际应用。对于这个实例，如果采用最先适宜策略，那么货物{4，2，3}存放在第一个集装箱，而{7，2}存放在第二个集装箱，{5，4}存放在第三个集装箱，{3，6}存放在第四个集装箱，而{2}存放在第五个集装箱。  
如果采用最优适宜策略，那么货物{4，2，4}存放在第一个集装箱，而{7，3}存放在第二个集装箱，{5，2，3 }存放在第三个集装箱，{6，2}存放在第四个集装箱。  
因为这两种方法都是采用的贪心策略，那么在一般情况下，是不能确保得到最优解的。

**试题答案**

（4）

【问题1】  
(1)j=0  
(2)b[j]=b[j]+s[i]  
(3) min= temp  
(4) b[m]= b[m]+s[i]  
【问题2】  
(5)贪心  
(6)贪心  
(7)O(n2)  
(8)O(n2)  
【问题3】  
(9)5  
(10)4  
(11)否

# **试题12(2012年上半年试题4)**

阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。  
【说明】  
    用两台处理机A和B处理n个作业。设A和B处理第i个作业的时间分别为ai和bi。由于各个作业的特点和机器性能的关系，对某些作业，在A上处理时间长，而对某些作业在B上处理时间长。一台处理机在某个时刻只能处理一个作业，而且作业处理是不可中断的，每个作业只能被处理一次。现要找出一个最优调度方案，使得n个作业被这两台处理机处理完毕的时间（所有作业被处理的时间之和）最少。  
算法步骤：  
    （1）确定候选解上界为R短的单台处理机处理所有作业的完成时间m，  
   IMG_276  
    （2）用p（x，y，k）=1表示前k个作业可以在A用时不超过x且在B用时不超过y时间  内处理完成，则p（x，y，k）=p（x-ak，y，k-1）||p（x，y-bk，k-1）（||表示逻辑或操作）。  
    （3）得到最短处理时问为min（max（x，y））。  
【C代码】  
  下面是该算法的C语言实现。  
（1）常量和变量说明  
  n: 作业数  
  m: 候选解上界  
  a: 数组，长度为n，记录n个作业在A上的处理时间，下标从0开始  
  b: 数组，长度为n，记录n个作业在B上的处理时间，下标从0开始  
  k: 循环变量  
  p: 三维数组，长度为（m+1）\*（m+1）\*（n+1）  
  temp: 临时变量  
  max: 最短处理时间  
  （2）C代码  
  #include<stdio.h>  
  int n， m;  
  int a[60]， b[60]， p[100][100][60];  
  void read(){ /\*输入n、a、b，求出m，代码略\*/}  
  void schedule(){ /\*求解过程\*/  
      int x，y，k;  
  for（x=0；x<=m；x++）{  
    for(y=0；y<m；y++）{  
   （1）  
        for（k=1；k<n；k++）  
    p[x][y][k]=0；  
  }  
  }  
  for（k=1；k<n；k++）{  
    for（x=0；x<=m；x++）{  
  for（y=0；y<=m；y++）{  
  if（x - a[k-1]>=0）    （2）    ；  
    if（ （3） ）p[x][y][k]=(p[x][y][k] ||p[x][y-b[k-1]][k-1]);  
        }  
    }  
    }  
  }  
  void write(){    /\*确定最优解并输出\*/  
  int x，y，temp，max=m;  
      for（x=0；x<=m；x++）{  
       for（y=0;y<=m;y++）{  
        if(  （4）  ）{  
    temp=（5） ；  
    if（temp< max）max = temp;  
        }  
       }  
     }  
    printf(“\n%d\n”，max），  
  }  
  void main(){read();schedule();write();}

【问题1】 （9分）  
根据以上说明和C代码，填充C代码中的空（1）～（5）。  
【问题2】（2分）  
根据以上C代码，算法的时间复杂度为（6）（用O符号表示）。  
【问题3】（4分）  
考虑6个作业的实例，各个作业在两台处理机上的处理时间如表4-1所示。该实例的最优解为（7），最优解的值（即最短处理时间）为（8）。最优解用（x1，x2，x3，x4，x5，x6）表示，其中若第i个作业在A上处理，则xi=l，否则xi=2。如（1，1，1，1，2，2）表示作业1，2，3和4在A上处理，作业5和6在B上处理

**表4-1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 作业1 | 作业2 | 作业3 | 作业4 | 作业5 | 作业6 |
| 处理机A | 2 | 5 | 7 | 10 | 5 | 2 |
| 处理机B | 3 | 8 | 4 | 11 | 3 | 4 |

**试题分析**

【问题1】  
下面我们来具体分析本试题。第（1）空所处的位置为schedule()函数的for循环中，从题目的描述和程序不难看出该三重循环的作用是给三维数组p赋初值，而根据题目描述可知数组k=0时，其对应的数组元素值都为1(因为这个时候没有作业，那么肯定可以在A用时不超过x且在B用时不超过y时间内处理完成)，因此第1空应该填p[x][y][0]=1。  
第（2）空在函数schedule()中的第二个三重for循环中，而且是在if结构下，只有if条件的结果为真时，才执行第（2）空的程序，从题目和程序也不难看出，这个三重for循环的作用就是要实现题目算法描述中的第（2）步，即求出p数组中各元素的值。那么当x - a[k-1]>=0为真时，即说明前k个作业可以在A用时不超过x内处理完成，那么根据题目意思，应该p（x，y，k）=p（x-ak，y，k-1），因此第（2）空的答案应该是p[x][y][k]=p[x-a[k-1]][y][k-1]。  
第（3）空if判定的条件表达式，根据条件为真后面执行的语句可以判定出，这里的条件是要判定是否前k个作业可以在B用时不超过y内处理完成，因此第（3）空的答案是y- b[k -1]>=0，其实本题与第（2）空可以参照来完成。  
第（4）空在函数write()中，是双重循环下if判定的条件，从题目注释来看，该函数是要确定最优解并输出的，那么结合该函数我们不难知识，确定最优解就是用这个双重循环来实现的，从前面的程序中，我们知道，所有的解的情况保存在数组p当中，那么现在就是要找出那个是最优解，其中max是用来存放当前最优解的，而临时变量temp要与max的值做一个比较，将较小的（当前最优）存放在max中，因此求最优解其实就是将所有解做一个比较，然后取出最优解。综上所述，再结合程序，我们不难知道第（4）空的答案是p[x][y][n]==1或者类似的表达式，p[x][y][n]==1表示当前情况下有一个解，那么这个解是x还是y呢？这还需要接着判定x与y的值谁更小，将更小的赋值给临时变量temp，因此第5空答案为(x>=y)?x:y。  
【问题2】  
本题主要考查时间复杂度，相对于第一问来说，要简单很多。从给出的程序来看，最高的循环是三重循环，因此其时间复杂度为O(m2n)或者O(n3)。  
【问题3】  
在本题给出的实例中，如果我们用题目描述的方式来求解，其过程也是相当复杂，因为在题目描述的情况下，数组p的长度为（33+1）\*（33+1）\*（6+1），由于我们不是计算机，要计算出该数组中各元素，肯定也不容易。在这种情况下，因为题目给出的作用只有6个，因此可以采用观察法，不难发现，本题最优解的值为15，最优解为(1，1，2，2，1，1)或者(2，1，2，1，2，2)。

**试题答案**

（4）

【问题1】  
(1)p[x][y][0]=1  
(2)p[x][y][k]=p[x-a[k-1]][y][k-1]  
(3)y- b[k -1]>=0  
(4)p[x][y][n]==1或p[x][y][n]或p[x][y][n]!=0  
(5)(x>=y)?x:y  
【问题2】  
(6) O(m2n)  
【问题3】  
(7)(1，1，2，2，1，1)  
(8)15

# **试题13(2011年上半年试题4)**

阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。  
【说明】  
    某应用中需要对100000个整数元素进行排序，每个元素的取值在0～5之间。排序算法的基本思想是：对每一个元素x，确定小于等于x的元素个数(记为m)，将x放在输出元素序列的第m个位置。对于元素值重复的情况，依次放入第m-l、m-2、…个位置。例如，如果元素值小于等于4的元素个数有10个，其中元素值等于4的元素个数有3个，则4应该在输出元素序列的第10个位置、第9个位置和第8个位置上。算法具体的步骤为：  
步骤1：统计每个元素值的个数。  
步骤2：统计小于等于每个元素值的个数。  
步骤3：将输入元素序列中的每个元素放入有序的输出元素序列。  
【C代码】  
下面是该排序算法的C语言实现。  
(1)常量和变量说明  
R: 常量，定义元素取值范围中的取值个数，如上述应用中R值应取6  
i：循环变量  
n：待排序元素个数  
a：输入数组，长度为n  
b：输出数组，长度为n  
c：辅助数组，长度为R，其中每个元素表示小于等于下标所对应的元素值的个数。  
(2)函数sort  
1    void sort(int n，int a[]，int b[]){  
2       int c[R]，i；  
3   for (i=0；i<    (1)  ：i++){  
4     c[i]=0；  
5       }  
6       for(i=0；i<n；i++）{  
7     c[a[i]] =   (2)  ；  
8       }  
9   for(i=1；i<R；i++）{  
10    c[i]=  （3）  
11      }  
12  for(i=0；i<n；i++）{  
13    b[c[a[i]]-1]=  (4)   ；  
14    c[a[i]]=c[a[i]]-1；  
15      }  
16    }

【问题1】（8分）  
     根据说明和C代码，填充C代码中的空缺(1)～(4)。  
【问题2】（4分）  
根据C代码，函数的时间复杂度和空间复杂度分别为 (5) 和 (6) （用O符号表示）。  
【问题3】（3分）     
     根据以上C代码，分析该排序算法是否稳定。若稳定，请简要说明（不超过100字）；若不稳定，请修改其中代码使其稳定（给出要修改的行号和修改后的代码）。

**试题分析**

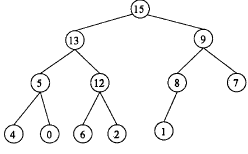
【问题1】  
    本题考查排序的相关内容。  
    题目告诉我们排序算法的基本思想是：对每一个元素x，确定小于等于x的元素个数(记为m)，将x放在输出元素序列的第m个位置。对于元素值重复的情况，依次放入第m-1、m-2、…的位置。而且题目告诉我们算法的步骤。  
    下面我们来具体分析本试题。第（1）空所处的位置为函数sort()中第一个for循环中，从题目的描述和程序不难看出该循环的作用是给数组c赋初值，而根据题目描述可知数组c是一个辅助数组，长度为R，因此第一空应填R。  
    第（2）空在函数sort()中的第二个for循环中，很显然第（2）空是给数组c赋值，而且其下标为数组a的相应的元素值。再根据题目的描述“c数组中每个元素表示小于等于下标所对应的元素值的个数”，很显然，这个for循环的作用是统计每个元素值的个数，因此第（2）空的答案应该是c[a[i]]+1。  
    第（3）空在第三个for循环中，而且第（3）空是给出数组c赋值，根据题目提供的算法的步骤，我们可知，这个时候应该要统计小于等于每个元素值的个数，而等于的元素个数记录在c[i]中，小于的元素个数记录在c[i-1]中，因此第（3）空的答案是c[i]+ c[i-1]。  
    第（4）空在最后一个for循环中，按题目要求，我们可以知道该for循环应该完成剩余的步骤3，即将输入元素序列中的每个元素放入有序的输出元素序列。而第（4）空是给数组b赋值，题目告诉我们b是输出数组，而a是输入数组，那么应该是将a中的值赋值值b中，因此第（4）空的答案应该为a[i]。  
【问题2】  
    本题主要考查时间复杂度与空间复杂度的分析。  
    首先我们来看空间复杂度，空间复杂度是对一个算法在运行过程中临时占用存储空间大小的量度。在sort()函数中，声明了两个整型变量n和i（可忽略），两个整型数组b和c，而a不属于函数sort的临时空间，因此函数sort()的空间复杂度为O(n+R)，这里由于在本题中R的值为6，因此也可以忽略，所以答案也可以是O(n)。  
    接着我们来分析时间复杂度，时间复杂度是度量算法执行的时间长短，函数sort()中有四个循环，其中有两个循环n次，而另外两个分别循环R-1和R次，因此时间复杂度应该为O(n+R)，由于R的值为6，这里可以忽略，因此答案也可以是O(n)。  
【问题3】  
所谓稳定性是指两个关键字相等的元素在排序前后的相对位置不发生变化，一般来讲，只要排序过程中比较和移动操作发生在相邻的元素间，排序方法是稳定的。本题中的排序是不稳定的，可以修改第12行的for循环为：for(i=n-1；i>=0；i--){即可。

**试题答案**

（4）

【问题1】  
(1)R     
(2)c[a[i]]+1     
(3)c[i]+c[i -1]     
(4)a[i]     
【问题2】  
(5)O(n+R)或者O(n)或n或线性     
(6)O(n+R)或者O(n)或n或线性     
【问题3】  
不稳定。修改第12行的for循环为：for(i=n-1；i>=0；i--){ 即可。

# **试题14(2010年下半年试题4-6)**

阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。  
【说明】  
堆数据结构定义如下：  
对于n个元素的关键字序列{a1，a2，...,an}，当且仅当满足下列关系时称其为堆。  
IMG_277  
在一个堆中，若堆顶元素为最大元素，则称为大顶堆；若顶堆元素为最小元素，则称为小顶堆。堆常用完全二叉树表示，图4-1是一个大顶堆的例子。  
  
**图4-1  大顶堆示例**  
堆数据结构常用于优先队列中，以维护由一组元素构成的集合。对应于两类堆结构，优先队列也有最大优先队列和最小优先队列，其中最大优先队列采用大顶堆，最小优先队列采用小顶堆。以下考虑最大优先队列。  
假设现已建好大顶堆A，且已经实现了调整堆的函数heapify（A,N,INDEX）。  
下面将C代码中需要完善的三个函数说明如下：  
（1）heapMaximum(A)：返回大顶堆A中的最大元素。  
（2）heapExtractMax(A)：去掉并返回大顶堆A的最大元素，将最后一个元素“提前”到堆顶位置，并将剩余元素调整成大顶堆。  
（3）maxHeapInsert(A,key)：把元素key插入到大顶堆A的最后位置，再将A调整成大顶堆。  
优先队列采用顺序存储方式，其存储结构定义如下：  
#define PARENT(i)   i/2  
typedef struct array{  
int\*int\_array;  //优先队列的存储空间首地址  
int array\_size;  //优先队列的长度  
int capacity;  //优先队列存储空间的容量  
}ARRAY;

 【C代码】

（1）函数heapMaximum  
int heapMaximum(ARRAY\*A){  return  (1)  ;  }  
（2）函数heapExtractMax  
int heapExtractMax(ARRAY\*A){  
int max;  
max=A->int\_array[0];  
（2）;  
A->array\_size --;  
heapify(A,A->array\_size,0);  //将剩余元素调整成大顶堆  
return max;  
}  
（3）函数maxHeapInsert  
int maxHeapInsert(ARRAY \*A,int key){  
int  i,\*p;  
if  (A->array\_size == A->capacity)    {  //存储空间的容量不够时扩充空间  
p=(int\*)realloc(A->int\_array,A->capacity \*2 \* sizeof(int));  
if (!p) return -1;  
A->int\_array = p;  
A->capacity = 2 \* A->capacity;  
}  
A->array\_size ++;  
i = （3） ;  
while (i > 0 && （4）){  
A->int\_array[i] = A->int\_array[PARENT(i)];  
i = PARENT(i);  
}  
（5）;  
return 0;  
}

**试题分析**

【问题1】  
    本题考查堆数据结构的相关内容。题目告诉我们函数heapMaximum（A）的功能返回大顶堆A中的最大元素；函数heapExtractMax（A）的功能是去掉并返回大顶堆A的最大元素，将最后一个元素“提前”到堆顶位置，并将剩余元素调整成大顶堆；而函数maxHeaplnsert（A， key）的功能是把元素key插入到大顶堆A的最后位置，再将A调整成大顶堆。  
第（1）空在函数heapMaximum（A）中，而且从程序中可以看出，是返回的结果，那么应该是大顶堆中最大元素，就应该是A->int\_array[0]。  
第（2）空在函数heapExtractMax（A）中，根据该函数的功能描述，并结合程序可以看出，第（2）空是在将最大元素移出后，那么接下了来应该处理将最后一个元素“提前”到堆顶位置，那么就应该是A->int\_array[0] = A->int\_array[A->array\_size -1]。  
第（3）（4）（5）空都在函数maxHeaplnsert（A， key）中。从程序和函数的功能我们可以知道，从程序第（3）空最后，其作用是找到元素key的插入位置并插入该元素。第（3）空是给变量i赋值，从后面的程序中我们可以看出i是做为数组下标的；而查找元素插入的位置应该从后往前的顺序，因此i的初值应该为A->array\_size – 1，从循环中也可以看出i的值在逐渐变小。  
第（4）空是循环的一个条件，而循环的作用是找到合适的插入位置，由于大顶堆的特点是根节点的值大于左右子树节点上的值，那么找到比待插入元素大的父节点时，应该就找到了它插入的合适位置，而每次操作后i的值被赋值为PARENT（i），很显然这是找到其父节点的存储位置，因此循环结束的一个条件就是找到一个比key值大的父节点，那么循环继续的条件就是父节点的值小于key的值，所以本空的答案为A->int\_array[PARENT(i)] <key。  
第（5）空就是插入元素，所以应该填A->int\_array[i] = key。  
【问题2】  
    根据题目描述，heapMaximum用来返回大顶堆A中的最大元素，而且大顶堆已经建成，只需要通过一步操作就能取到。因此时间复杂度是O（1），  
    而对于heapExtractMax是用来去掉大顶堆A的根，然后重新建堆，当输出堆顶结点并将堆中最后一个结点设置为根结点之后，根结点将有可能不再满足堆的性质，所幸的是整个序列也只有根结点一处的堆结构可能被破坏，其余结点仍然满足堆性质，故可利用性质进行堆调整，算法的基本思想为：将新堆顶沿着其关键字较大的孩子结点向下移动，直到叶子结点或者满足堆性质为止。因此相对于有N个元素的堆，只需要log2n次比较即可完成，因此时间复杂度是O（log2n），这与书本说堆排序的算法时间复杂度是：O(nlog2n)不冲突，因为书本上是对堆中所有元素进行操作，而这里其实相当于只将一个元素入堆，因此少了一个n。同样的道理可以得到maxHeaplnsert的时间复杂度O（log2n）。  
【问题3】  
这个我们可以结合题目给出的那个大顶堆的图来看，首先将key插入在最后，应该是8这个节点的右子树，由于10比8大，所以应该互换，再与节点9比较，由于10任然大于9，所以也应该互换，这个时候再与其父节点15比较，由于小于15，所以不需要再调整，那么调整后的结果就是10这个元素应该作为根节点15的右子树。那么很显然10应该是在堆A中第3个位置。

**试题答案**

（4）【问题1】  
（1）A->int\_array[0]  
（2）A->int\_array[0]= A->int\_array[A->array\_size-1]  
（3）A-> array\_size-1  
（4）A->int\_array[PARENT(i)]<key  
（5）A->int\_array[i]=key  
【问题2】  
（6）O(1)   （7）O(lgn)    （8）O(lgn)  
【问题3】  
（9）3